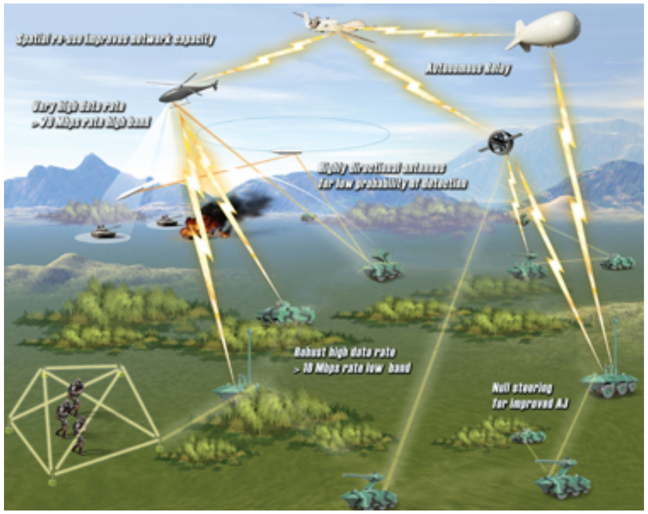
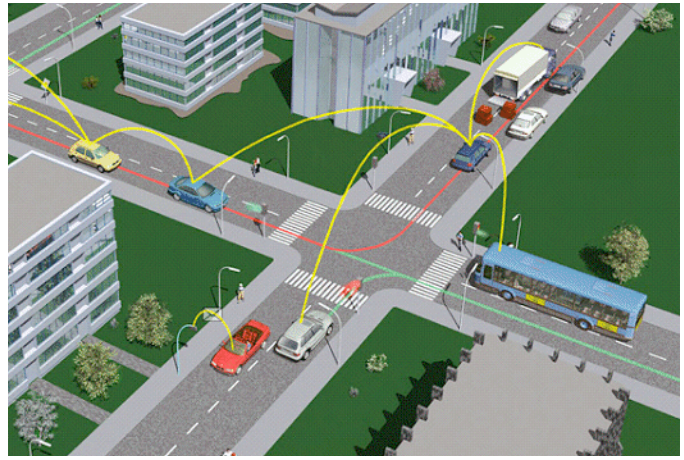
MURI Review
October 2008
Caltech

Verification and Performance in Networked Embedded Systems

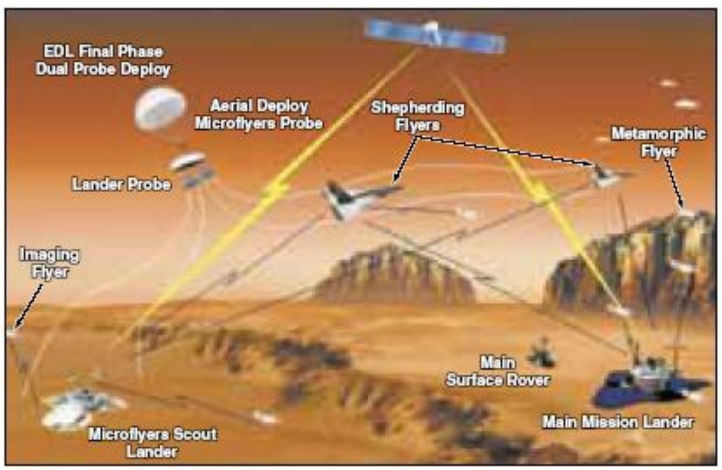
Eric Klavins (University of Washington)
J.M. McNew (Toyota Research)
David Thorsley (University of Washington)
Steve Safarik (University of Washington)

Setting

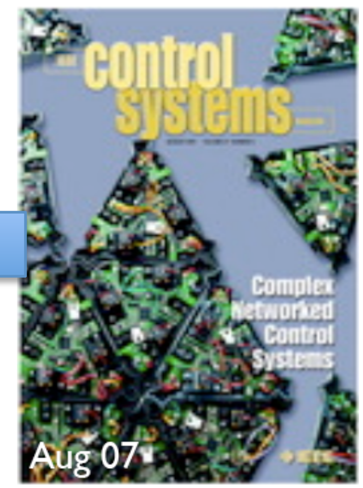




- Heterogeneity
- Local communication
- Motion
- Changing environment
- Stochasticity
- Complex Tasks / Subtasks

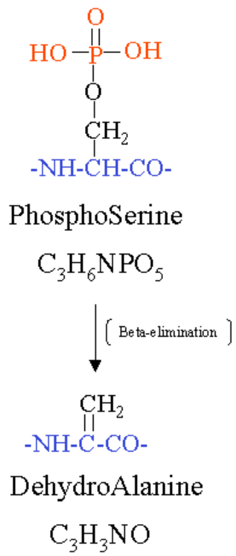


Metabolobtics

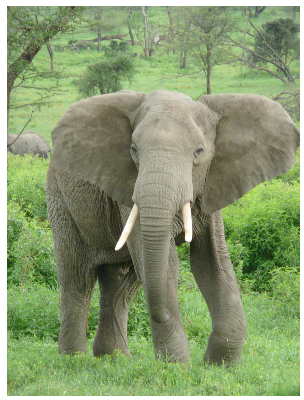


Programmable Parts

Challenges



?



- Abstraction
- Programming and design
- Specification
- Verification
- Performance

```
Send(strTestMessage.GetBuffer(),strTestMessage.GetLength()+1)
```

?

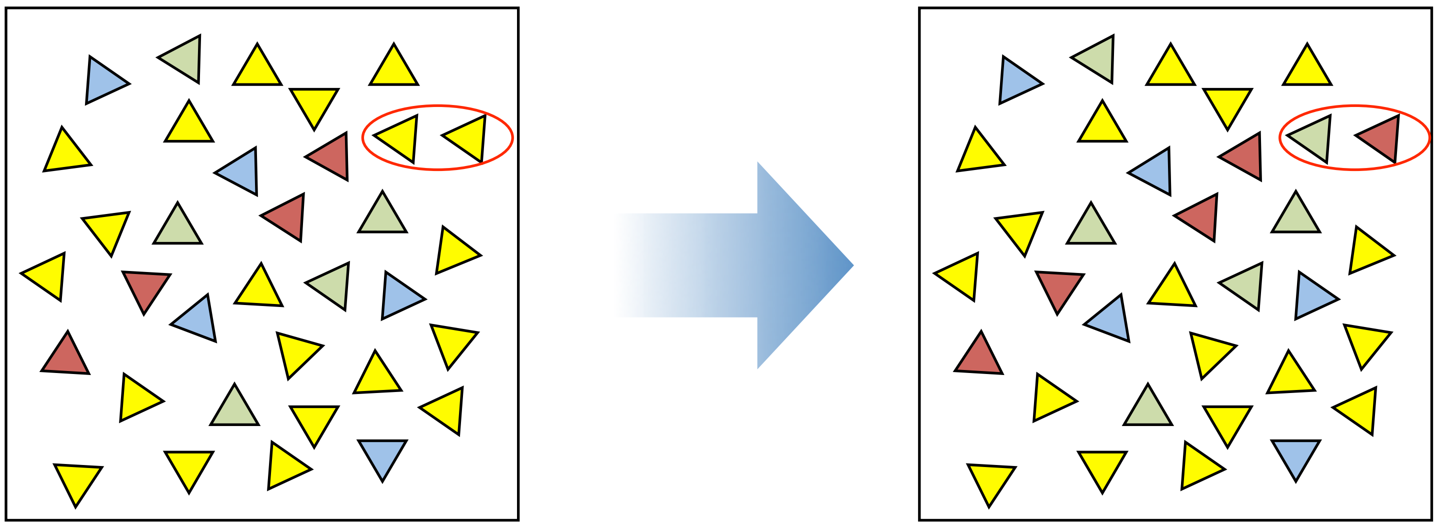
Overview of UW Progress

1. Testbeds: PPT, EFRI
2. Suite of examples specified and verified
3. Lyapunov function co-design
4. Separation of continuous and discrete design
5. Separation of verification and performance tuning
6. Pseudometrics to compare, diagnose and optimize stochastic systems
7. Diagnosis / observation of stochastic process with low-dimensional, stochastic output

This talk (Overview of McNew's Thesis)

David's talk

Abstraction: Concurrency

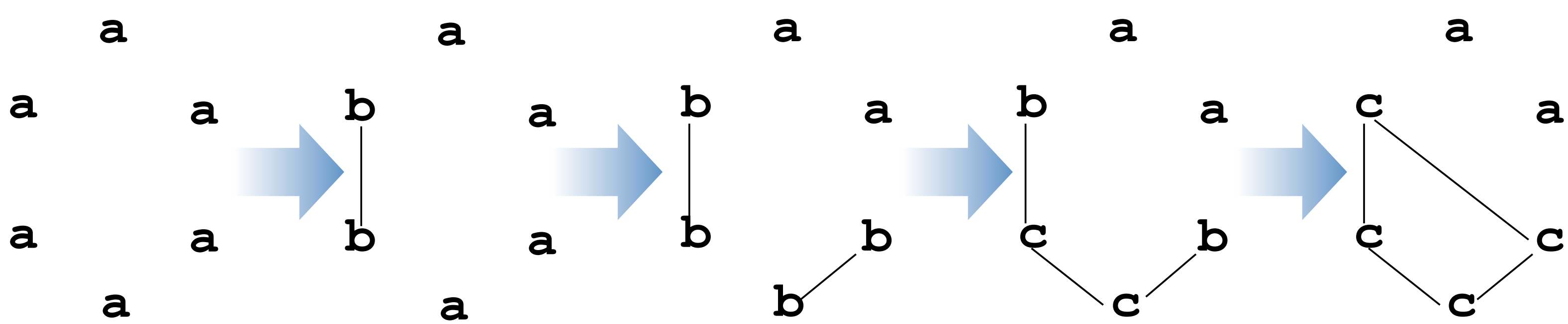


A concurrent program:
Each guards and rule can be evaluated by small groups of agents without global knowledge.

$$\left\{\begin{array}{l} guard_1 : rule_1 \\ guard_2 : rule_2 \\ . \\ . \\ . \\ guard_n : rule_n \end{array}\right.$$

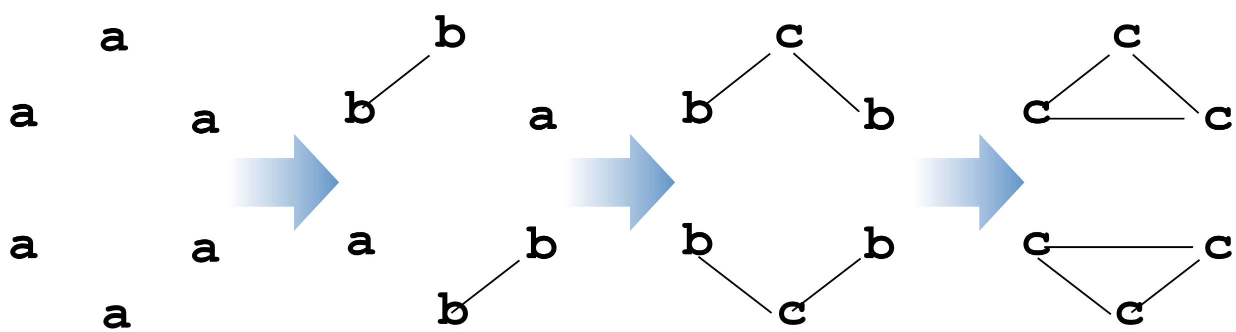
Non-determinism: Which rule? Where to apply it? When to apply it?
Abstraction: How is the rule applied? – That's an implementation issue!

Example: Graph Grammars



$$
\Phi = \begin{cases} a \quad a \quad \rightharpoonup \quad b - b, \\ a \quad b \quad \rightharpoonup \quad b - c, \\ b \quad b \quad \rightharpoonup \quad c - c \end{cases}
$$

In general, each node carries a data structure, and rules operate on pairs of data structures.



E. Klavins, R. Ghrist and D. Lipsky, A Grammatical Approach to Self-Organizing Robotic Systems, IEEE Transactions on Automatic Control. Jun. 2006, Vol. 51, No. 5, pp. 949-962.

E. Klavins, Programmable Self-Assembly. Control Systems Magazine, Aug. 2007. Vol. 24, No. 4, pp. 43-56.

Graph Grammar Rules More Generally

Rule name

Unbound node indices

Existential variable

Precondition on data structures maintained by i, j and k

Precondition local network

Postcondition on network
(there may also be postconditions on data)

Rule r_3 *(Merge Branches)*

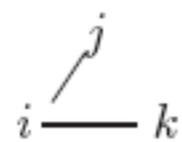
Vertices : i, j, k

Variables : b_i

Precondition:

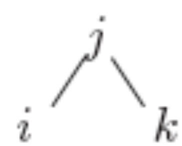
$$i.mode = t \wedge j.mode = a \wedge k.mode = a$$
$$ij.order(j) < b_i \wedge ik.order(k) < b_i$$

$E \; =$



Effect:

$E \; :=$

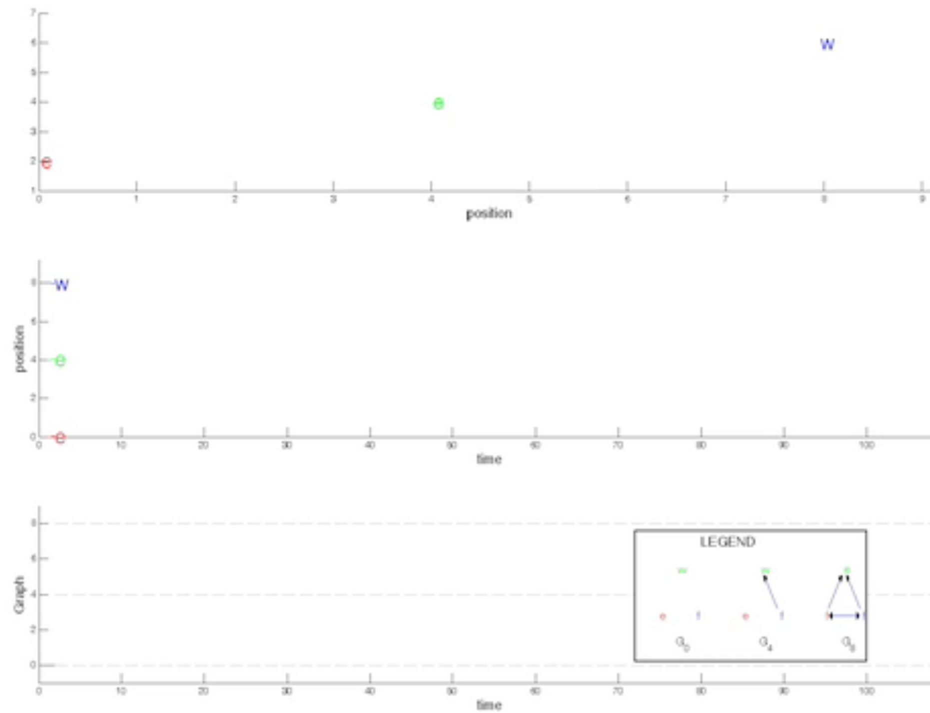


Example: Embedded Graph Grammars

$$\mathbf{u}_i(\gamma) = \begin{cases} 1 & \text{if } i.mode = \mathtt{east} \\ -1 & \text{if } i.mode = \mathtt{west} \\ \sum_{j \in N(i)} \mathbf{x}_j - \mathbf{x}_i & \text{if } i.mode = \mathtt{follow} \end{cases}$$

Φ:

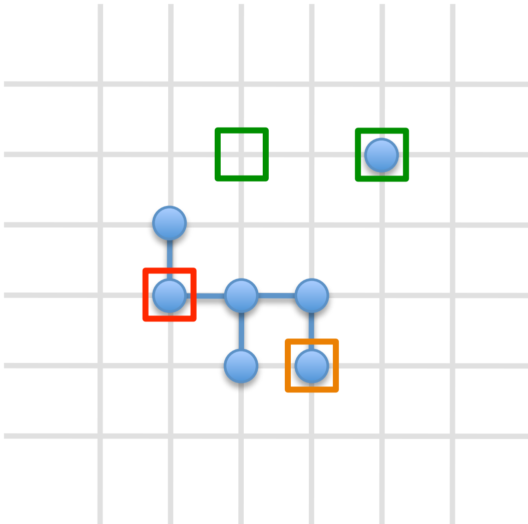
Rule r_1 (Join)	Rule r_2 (Close Cycle)
Precondition: $\|\|x_i - x_j\|\| < 3$, $e \quad w$ **Effect:** $w \leftarrow f$	**Precondition:** $\|\|x_i - x_j\|\| < 5 \wedge \|\|x_k - x_j\|\| < 5 \wedge \|\|x_i - x_k\|\| < 5$ e $w \leftarrow f$ **Effect:** $f \rightarrow e$, $f \rightarrow f$, $e \leftarrow f$



Animation

J. M. McNew. Ph.D. Thesis. 2008.
J. M. McNew, E. Klavins and M. Egerstedt. Solving Coverage Problems with Embedded Graph Grammars. HSCC 2007.

Example: Metabolobotics



NSF EFRI: Build it and program it!
MURI: Verification tools should apply!

- Bond
- Part
- Welder
- Breaker
- Carrier

Action name: $\text{manipulate}(i, j):$

Least Specific Guard: $(r_i = CARRIER \vee r_i = BONDER \vee r_i = BREAKER) \wedge (x_i, y_i) = (x_j, y_j)$

Update: $m_i' = m_i \cup j$

Other actions: move, weld, break, label, communicate

Example: Metabolobotics

role == Carrier ∧ (¬ EjiMj ∧ ¬ Efree0) : MoveU[i]
role == Carrier ∧ (¬ EjiMj ∧ ¬ Efree0) : MoveD[i]
role == Carrier ∧ (¬ EjiMj ∧ ¬ Efree0) : MoveL[i]
role == Carrier ∧ (¬ EjiMj ∧ ¬ Efree0) : MoveR[i]

role == Carrier ∧ (¬ EjiMj ∧ Efree0) :
  Manipulate[i, iFree] ∧
  WriteSublabel[Last[PartsGetManipulating[i]], idxLabelName, nameCarried]

role == Carrier ∧ (EjiMj ∧ ¬ EfinalD1) : MoveU[i]
role == Carrier ∧ (EjiMj ∧ ¬ EfinalD1) : MoveD[i]
role == Carrier ∧ (EjiMj ∧ ¬ EfinalD1) : MoveL[i]
role == Carrier ∧ (EjiMj ∧ ¬ EfinalD1) : MoveR[i]

role == Carrier ∧ (EjiMj ∧ EfinalD1) :
  WriteSublabel[Last[PartsGetManipulating[i]], idxLabelName, nameFinal] ∧
  Release[i]

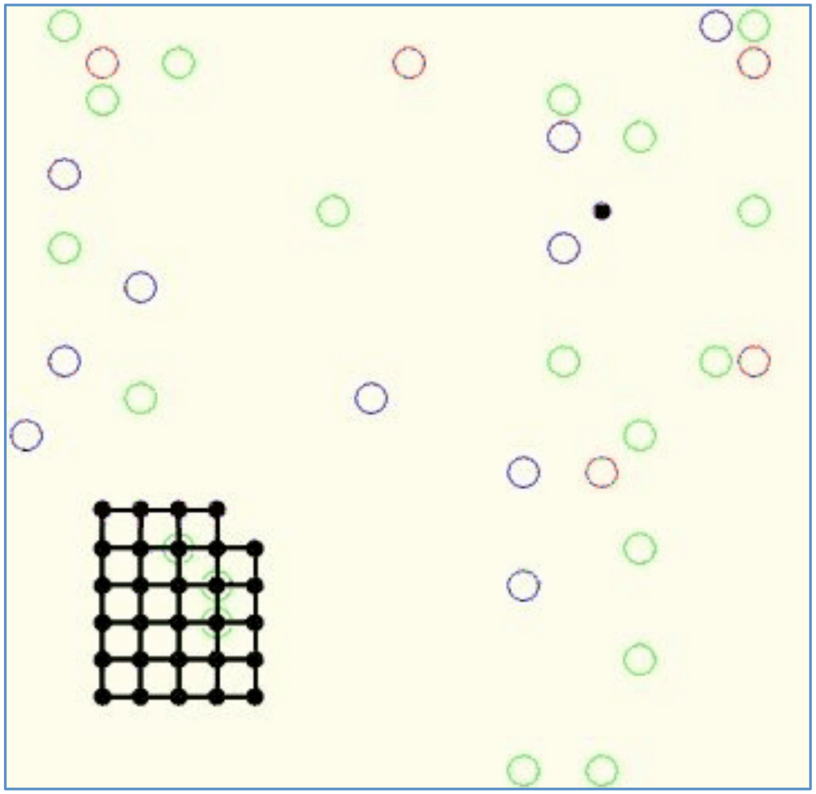
role == Welder ∧ (¬ EfinalD0 ∨ ¬ EfinalD1 ∨ BondedQ[iFinal0, iFinal1]) : MoveU[i]
role == Welder ∧ (¬ EfinalD0 ∨ ¬ EfinalD1 ∨ BondedQ[iFinal0, iFinal1]) : MoveD[i]
role == Welder ∧ (¬ EfinalD0 ∨ ¬ EfinalD1 ∨ BondedQ[iFinal0, iFinal1]) : MoveL[i]
role == Welder ∧ (¬ EfinalD0 ∨ ¬ EfinalD1 ∨ BondedQ[iFinal0, iFinal1]) : MoveR[i]

role == Welder ∧ EfinalD0 ∧ EfinalD1 ∧ ¬ BondedQ[iFinal0, iFinal1] :
  Manipulate[i, iFinal0] ∧
  Bond[i, iFinal0, iFinal1] ∧
  Release[i]

role == Breaker ∧ ¬ Einitial0 : MoveU[i]
role == Breaker ∧ ¬ Einitial0 : MoveD[i]
role == Breaker ∧ ¬ Einitial0 : MoveL[i]
role == Breaker ∧ ¬ Einitial0 : MoveR[i]

role == Breaker ∧ Einitial0Unbonded :
  Manipulate[i, iInitial0Unbonded] ∧
  WriteSublabel[iInitial0Unbonded, idxLabelName, nameFree] ∧
  Release[i, iInitial0Unbonded]

role == Breaker ∧ Einitial0Bonded :
  Manipulate[i, iBondedPair[[1]]] ∧
  Break[i, iBondedPair[[1]], iBondedPair[[2]]] ∧
  Release[i, iBondedPair[[1]]]

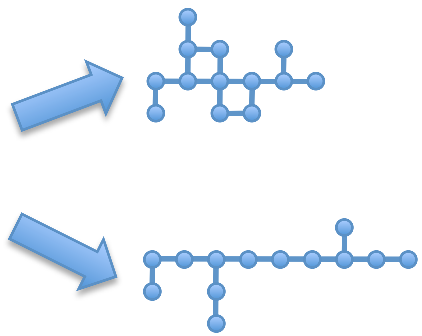


Animation

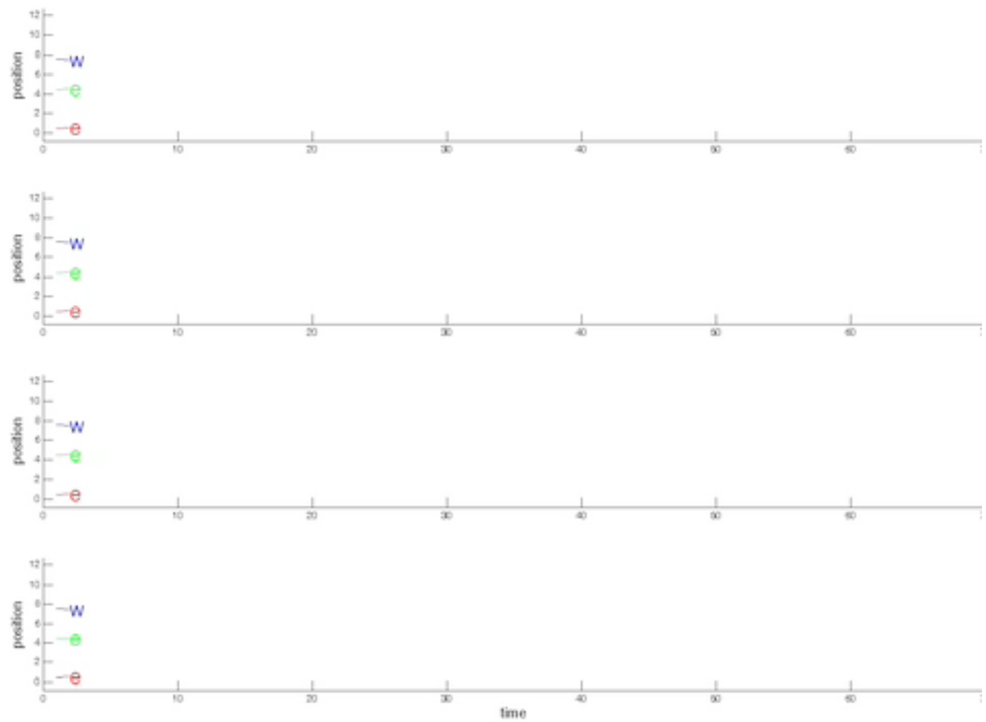
Sources of Nondeterminism (Due to Abstraction)

- Rule application order
- Rule application time
- Rule application place

Initial condition



Metabolobotics example produces many different structures (in many different ways) – but they are (a) connected and (b) contain all parts.



EGG example produces different results depending on timing and order.

Animation

Specification: Make statements about desired <u>sets</u> of possible trajectories.
Verification: Check whether our programs always produce trajectories contained within those sets.

Verification: States, Updates and Trajectories

$\sigma = \langle s_0, s_1, ... \rangle$ (or $\sigma : \mathbb{R}^+ \cup \{0\} \rightarrow S$)

$s[\![f]\!] \Leftrightarrow$ *s* is a state contained in the set of states specified by the formula *f*.

$s[\![a]\!]t \Leftrightarrow$ *(s,t)* is a pair of states contained in the set of pairs specified by the action *a*.

$\sigma[\![F]\!] \Leftrightarrow$ σ is a trajectory contained in the set of trajectories specified by the formula F.

Lamport’s Temporal Logic of Actions

(Lamport, ACM Toplas 16, 3 (May 1994) 872-923)

Semantics

$$s[\![f]\!] \triangleq f(\forall \text{`}v\text{'} : s[\![v]\!]/v) \qquad \sigma[\![F \wedge G]\!] \triangleq \sigma[\![F]\!] \wedge \sigma[\![G]\!]$$

$$s[\![\mathcal{A}]\!]t \triangleq \mathcal{A}(\forall \text{`}v\text{'} : s[\![v]\!]/v,\ t[\![v]\!]/v') \qquad \sigma[\![\neg F]\!] \triangleq \neg\sigma[\![F]\!]$$

$$\models \mathcal{A} \triangleq \forall s, t \in \mathbf{St} : s[\![\mathcal{A}]\!]t \qquad \models F \triangleq \forall \sigma \in \mathbf{St}^{\infty} : \sigma[\![F]\!]$$

$$s[\![Enabled\ \mathcal{A}]\!] \triangleq \exists t \in \mathbf{St} : s[\![\mathcal{A}]\!]t$$

$$\langle s_0, s_1, \ldots \rangle[\![\Box F]\!] \triangleq \forall n \in \mathsf{Nat} : \langle s_n, s_{n+1}, \ldots \rangle[\![F]\!]$$

$$\langle s_0, s_1, \ldots \rangle[\![\mathcal{A}]\!] \triangleq s_0[\![\mathcal{A}]\!]s_1$$

Additional notation

$$p' \triangleq p(\forall \text{`}v\text{'} : v'/v) \qquad \Diamond F \triangleq \neg\Box\neg F$$

$$[\mathcal{A}]_f \triangleq \mathcal{A} \vee (f' = f) \qquad F \leadsto G \triangleq \Box(F \Rightarrow \Diamond G)$$

$$\langle \mathcal{A} \rangle_f \triangleq \mathcal{A} \wedge (f' \neq f) \qquad \mathrm{WF}_f(\mathcal{A}) \triangleq \Box\Diamond\langle \mathcal{A} \rangle_f \vee \Box\Diamond\neg Enabled\ \langle \mathcal{A} \rangle_f$$

$$Unchanged\ f \triangleq f' = f \qquad \mathrm{SF}_f(\mathcal{A}) \triangleq \Box\Diamond\langle \mathcal{A} \rangle_f \vee \Diamond\Box\neg Enabled\ \langle \mathcal{A} \rangle_f$$

Guard:Rule pairs, programs, refinements, implementations and specification can all be written in TLA.

Verification: Progress via Lyapunov Co-Design

Lamport's TLA inference rule for progress uses a *discrete Lyapunov function* H_c.

Initial Condition **Discrete Lyapunov Function** **Goal Condition**

$$\frac{F \wedge (\mathsf{c} \in \mathsf{S}) \;\Rightarrow\; (H_{\mathsf{c}} \leadsto (G \vee \exists \mathsf{d} \in \mathsf{S} : (\mathsf{c} \succ \mathsf{d}) \wedge H_{\mathsf{d}}))}{F \;\Rightarrow\; ((\exists \mathsf{c} \in \mathsf{S} : H_{\mathsf{c}}) \leadsto G)}$$

McNew's Lyapunov Codesign Method:

1) Build individual behaviors, each decreasing it's own LF U_i
2) Compose behaviors, noting priority
3) Verify that U_i increasing -> U_j decreases for some j<I
4) $U=(U_1,\ldots,U_k)$ is a Lyapunov function under the lexicographic ordering

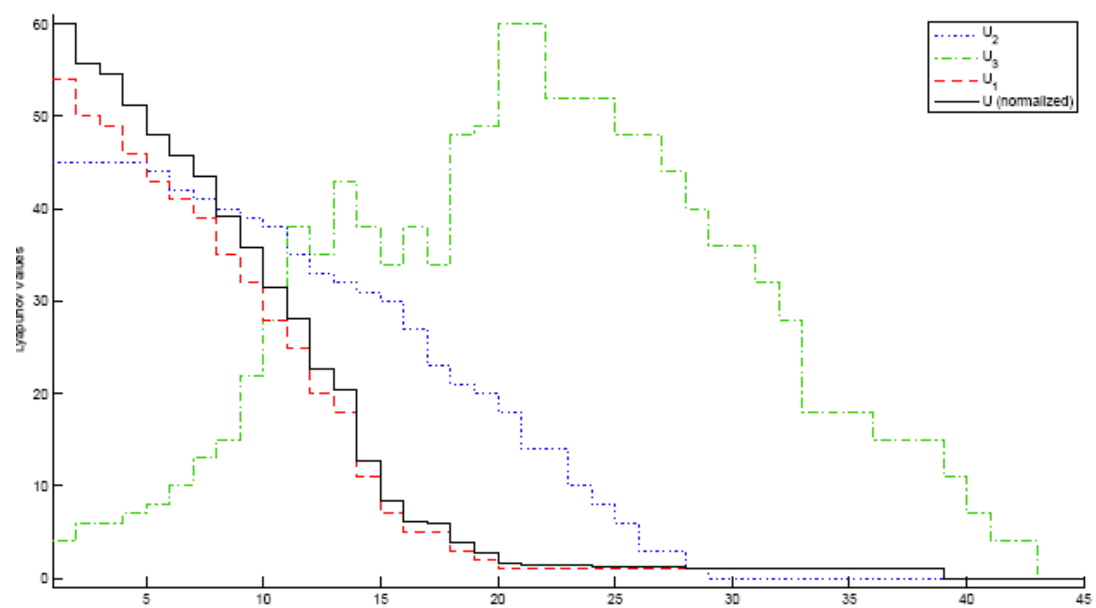
Verification: Progress via LLFs

$$\Phi = \begin{cases} (w,x) \quad (y,z) \rightharpoonup (w-1,x) \quad (y,z+x) \\ (0,x) \quad (y,z) \rightharpoonup (0,x) - (y,z) \\ (0,x) - (y,z) \rightharpoonup (0,x) - (\max(y-1,0), max(z-1,0)) \end{cases}$$

Lexicographic
Lyapunov
Function

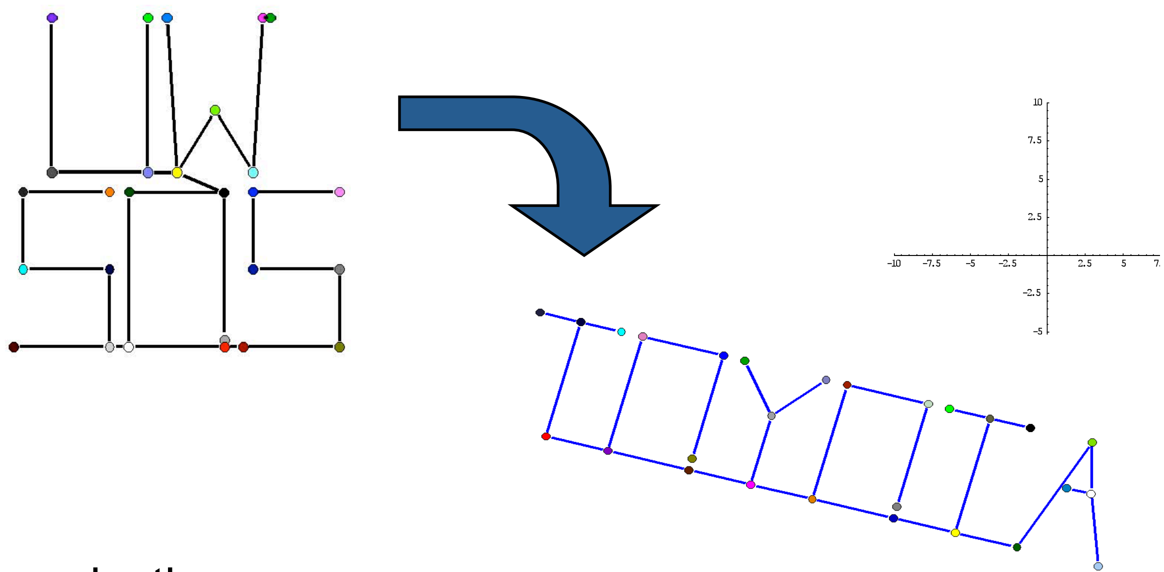
Let $\mathbf{U} = (\mathbf{U}_1, \mathbf{U}_2, \mathbf{U}_3)$ where

1. $\mathbf{U}_1 = \sum_{i \in V} i.a$
2. $\mathbf{U}_2 = (|V|^2 - |V|)/2 - |\mathrm{E}|$
3. $\mathbf{U}_3 = \sum_{i \in V} i.b$



This grammar results in a fully connected graph all of whose vertices are labeled (0,0).

Tree-tree Reconfiguration



Assumptions:

- Local communication.
- Unknown initial state.

Goals:

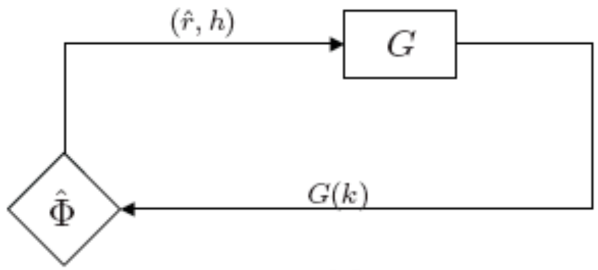
- Convergence to an isomorphism of the desired tree formation (progress).
- All intermediate states are tree formations (safety).
- Correct behavior from <u>all</u> initial conditions.

J. M. McNew and E. Klavins. Non-deterministic Reconfiguration of Tree Formations. American Control Conference, June 2008, pp. 690-697.

Verification: Separation of Discrete and Continuous

$$\begin{array}{c} f \\ \downarrow \\ \hat{f} \end{array}$$

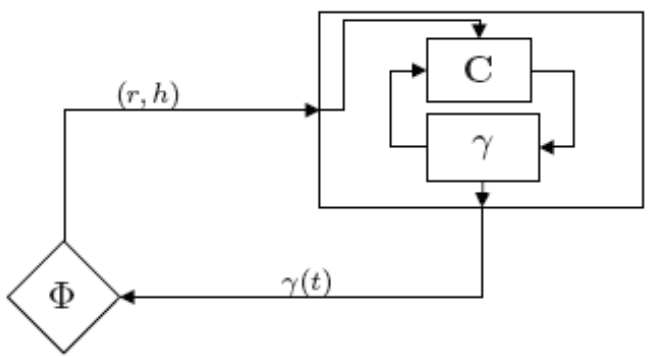
$(G_0, \hat{\Phi}) \models \hat{f}$



1. Write a program for only the discrete part of the problem (e.g. network reconfiguration).

$$\begin{array}{c} (f, \Phi) \\ \uparrow \\ (\hat{f}, \hat{\Phi}) \end{array}$$

$(\gamma_0, \Phi, C) \models f$

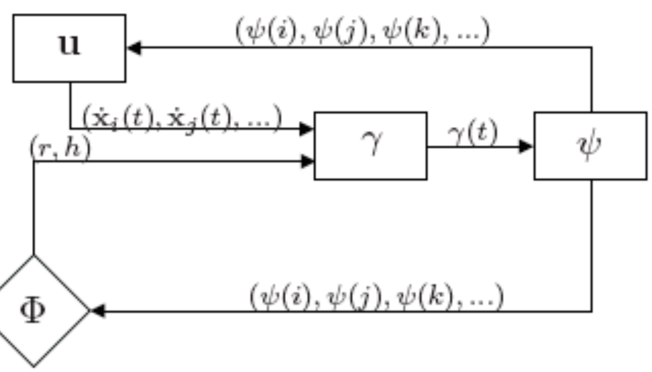


2a) Lift the discrete solution to the hybrid (discrete/continuous state).

2b) Determine a specification, C, that if met by u and ψ guarantees safety and progress in discrete solution.

$(\psi, \mathbf{u}) \models C$

$(\gamma_0, \Phi, \psi, \mathbf{u}) \models f$



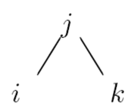
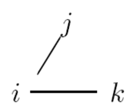
3) Design u and ψ to meet the specification C.

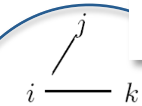
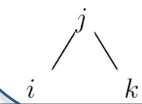
Tree-tree Reconfiguration

(Discrete Part)

Rule r_1
Vertices : i, j
Precondition:
$i.mode = t \wedge j.mode = a$
Denote $i.role$ by v. $\exists w \in N_{i.tree}(v)$ such that $(w.mode = a \wedge vw.order(w) = ij.order(j))$
$G[\{i,j\}] = i - j$
Effect:
$w.mode := t, \quad j.tree := i.tree$
$j.mode := t, \quad j.role := w$
$ij.offset := vw.offset, \quad ij.head := j$

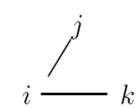
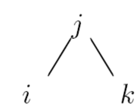
Each robot maintains its role and an estimate of the tree

Rule r_2
Vertices : i, j, k
Precondition:
$i.mode = t \wedge j.mode = a \wedge k.mode = a$
$ij.order(j) > b_i$
$jk.order(k) \geq b_i \vee ij.order(j) - jk.order(k) \geq b_i$
$G[\{i,j,k\}] =$ 
Effect:
$G[\{i,j,k\}] :=$ 

Rule r_3
Vertices : i, j, k
Precondition:
$i.mode = t \wedge j.mode = a \wedge k.mode = a$
$ij.order(j) < b_i \wedge ik.order($
$G[\{i,j,k\}] =$ 
Effect:
$G[\{i,j,k\}] :=$ 

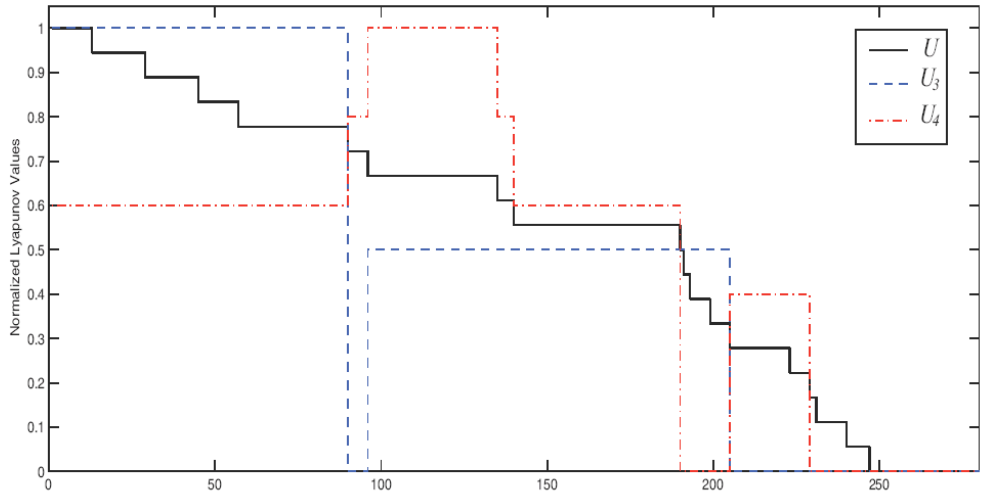
Local network reconfigurations are guaranteed to preserve tree

Rule r_4
Vertices : i, j, k
Precondition:
$i.mode = t \wedge j.mode = a \wedge k.mode = a$
$ij.order(j) > b_i$
$jk.order(k) < b_i \wedge ij.order(j) - jk.order(k) < b_i$
$G[\{i,j,k\}] =$ 
Effect:
$i.mode := m$

Rule r_5
Vertices : i, j, k
Precondition:
$i.mode = m \wedge j.mode = a \wedge k.mode = a$
$G[\{i,j,k\}] =$ 
Effect:
$G[\{i,j,k\}] :=$ 
$i.mode := a$

J. M. McNew and E. Klavins. Non-deterministic Reconfiguration of Tree Formations. American Control Conference, June 2008, pp. 690-697.

Verification: Progress



U_1—The number of vertices yet to be matched to the target graph.

U_2—The number of t-a edges to which r_2 might apply.

U_3—The average distance from b_i on t-a edges to which r_2 might apply.

U_4—The summed distance from b_i on t-a edges to which r_3 applies.

U_5—The number of subgraphs to which r_4 or r_5 apply.

U_6—The number of subgraphs to which r_5 applies.

J. M. McNew and E. Klavins. Non-deterministic Reconfiguration of Tree Formations. American Control Conference, June 2008, pp. 690-697.

Tree-tree Reconfiguration: Continuous Part

Intermediate specification on u and ψ

Safety: $\Box(ij \in E \rightarrow ij \in E_\psi)$

Progress: $\dot{x} = u \leadsto (\dot{x} \neq u \vee \exists r.x \in \text{cont\_guard}(r))$

Rule r_5 *(Merge Ahead)*

Vertices : i, j, k

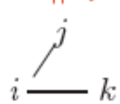
Variables : b_i

Precondition:

$i.mode = m \wedge j.mode = a \wedge k.mode = a$

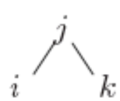
$||\mathbf{x}_i - \mathbf{x}_j|| < \Delta \wedge ||\mathbf{x}_i - \mathbf{x}_k|| < \Delta \wedge ||\mathbf{x}_k - \mathbf{x}_j|| < \Delta$

$E \ =$



Effect:

$E \ :=$



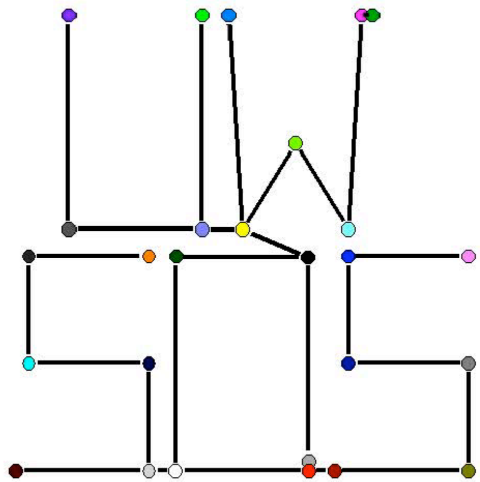
$i.mode := a$

$$\dot{\mathbf{x}}_i = -\sum_{j \in N_\psi(i)} \frac{2(\Delta - ||\alpha||) - ||\mathbf{x}_i - \mathbf{x}_j - \alpha||}{(\Delta - ||\alpha|| - ||\mathbf{x}_i - \mathbf{x}_j - \alpha||)^2}(x_i - x_j - \alpha)$$

Simple implementation (can prove it satisfies C)

Continuous predicate updates to rules.

Simulation



The Separation of Verification and Performance Tuning

Code for a *non-deterministic walk*

$$
\begin{aligned}
&mode = search : move\_lt \\
&mode = search : move\_rt \\
&mode = search : move\_up \\
&mode = search : move\_dn
\end{aligned}
$$

TLA spec with fairness on movement

$$
\begin{aligned}
\Pi \quad = \quad & \Box(x_{min} \leq x \leq x_{max} \wedge y_{min} \leq y \leq y_{max}) \\
= \quad & \bigwedge_{x_{min} < w \leq x_{max}} \mathrm{SF}(x = w \wedge move\_lt) \\
= \quad & \bigwedge_{x_{min} \leq w < x_{max}} \mathrm{SF}(x = w \wedge move\_rt) \\
= \quad & \bigwedge_{y_{min} < z \leq y_{max}} \mathrm{SF}(y = z \wedge move\_dn) \\
= \quad & \bigwedge_{y_{min} \leq w < y_{max}} \mathrm{SF}(y = z \wedge move\_up) \\
& \vdash \Diamond(x = x^* \wedge y = y^*)
\end{aligned}
$$

Inference rule

$$
\frac{\text{true}}{\text{mode} = \text{search} \leadsto (\text{mode} \neq \text{search} \vee (x = x^* \wedge y = y^*))}
$$

Safarik, Napp and Klavins, In progress.

The Separation of Verification and Performance Tuning

Code for a *feedback controlled random walk*

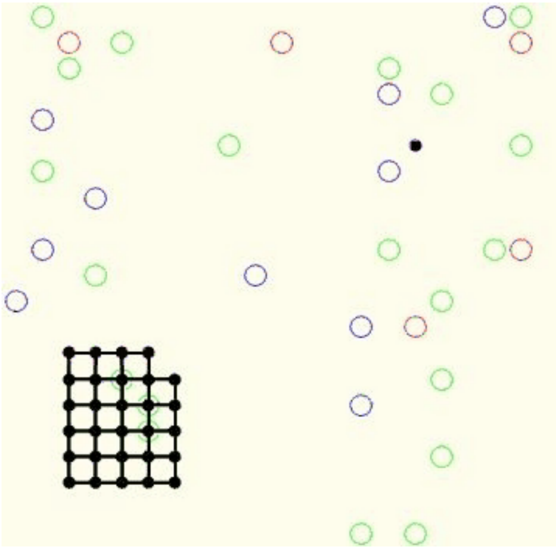
$$\begin{array}{ll} mode = search : move\_lt & u_l(x,y,t)dt \\ mode = search : move\_rt & u_r(x,y,t)dt \\ mode = search : move\_up & u_u(x,y,t)dt \\ mode = search : move\_dn & u_d(x,y,t)dt \end{array}$$

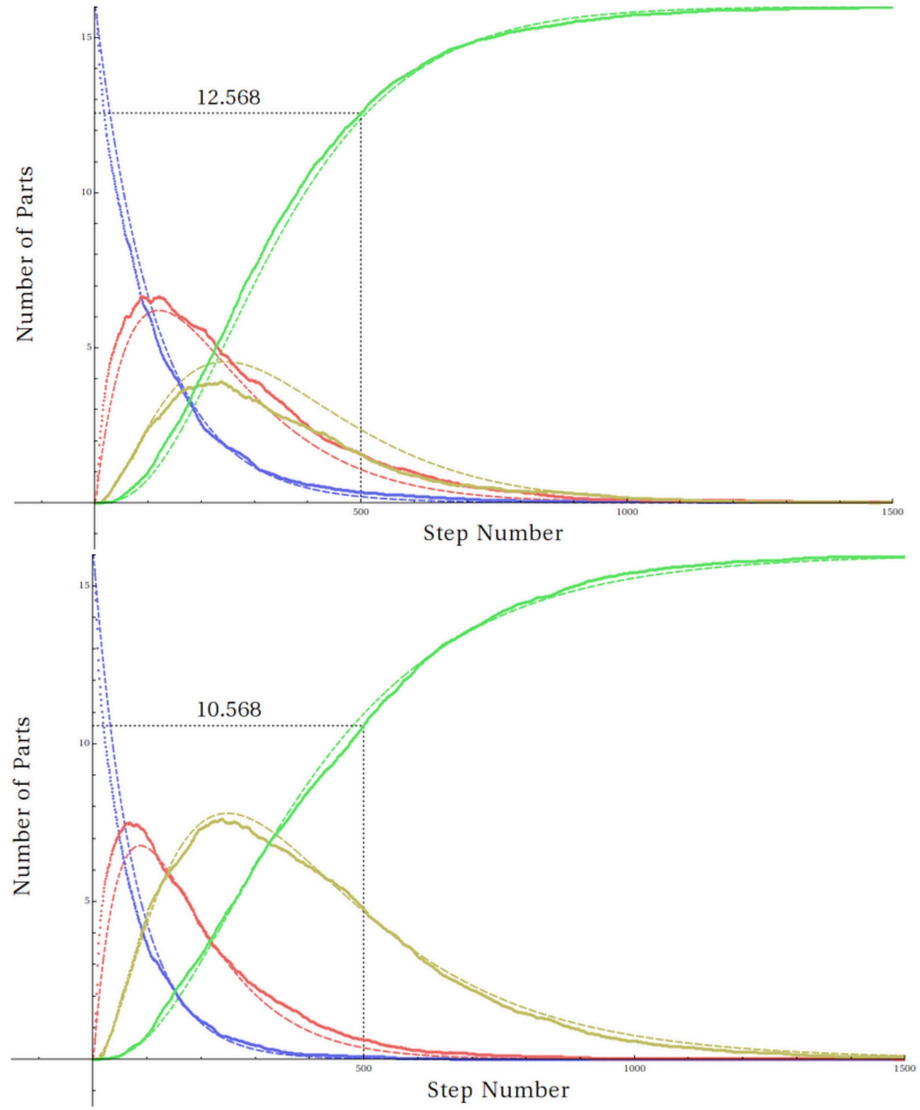
Probability that the rule will fire in the next dt seconds given the state is (x,y)

Thm: Set of unfair trajectories of the non-deterministic system has measure zero in any probabilistic implementation in which $u_*(x,y,t) > 0$ for all x, y and t.

Thus, even an (almost) deterministic search algorithm is guaranteed to be correct.

An Example: Tuning Metabolism While Preserving Correctness





Reduced model (k_i depends on allocation of welders, breakers, carriers).

$$B \xrightarrow{k_1} F$$
$$F \xrightarrow{k_2} P$$
$$P \xrightarrow{k_3} D$$

Safarik, Napp and Klavins, In progress.

Summary

- Examples/Testbeds
  – Next: EFRI testbed available / simulator enhanced
- Lyapunov Co-Design
  – Next: Lyapunov co-design in stochastic systems (e.g. Parrilo)
- Separation of continuous and discrete design
  – Next: Formalize in TLA
- Separation of correctness and performance
  – Next: Formalize in TLA
  – Next: Feedback controlled performance that does not affect correctness
- Stochastic diagnosis, model reduction
  – David's talk